UNIVERSIDADE DE UBERABA

YGOR GONZAGA JOSUÉ

DOCUMENTAÇÃO TÉCNICA DA PLATAFORMA INCUBEPRO

UBERABA

2025

IncubePro

IncubePro - Documentação Técnica

Visão Geral

A **IncubePro** é uma plataforma desenvolvida para impulsionar o financiamento e a evolução de projetos de software, com foco no fortalecimento da inovação tecnológica no Brasil.

Objetivos Principais

- Conectar desenvolvedores, investidores e mentores em um ambiente digital unificado
- Facilitar a publicação, gestão e apoio a projetos promissores
- Utilizar crowdfunding como ferramenta central de captação de recursos
- Validar publicamente ideias inovadoras, fortalecendo sua credibilidade
- Contribuir para a retenção de talentos e fortalecimento do ecossistema de inovação

Fluxo da Plataforma

1. **Desenvolvedores** publicam seus projetos na plataforma
2. **Investidores e comunidade** descobrem e apoiam projetos promissores
3. **Mentores** oferecem orientação e suporte técnico
4. **Crowdfunding** viabiliza o financiamento coletivo das ideias
5. **Acompanhamento** transparente do progresso dos projetos

Arquitetura e Tecnologias

Stack Tecnológico

Backend (Node.js)

- **Express.js** - Framework web principal
- **MongoDB** - Banco de dados NoSQL (via MongoDB Atlas)
- **Mongoose** - ODM para MongoDB
- **JWT** - Autenticação baseada em tokens
- **EJS** - Template engine para páginas web

Dependências Principais

```
{
  "bcryptjs": "^2.4.3",           // Criptografia de senhas
  "cookie-parser": "^1.4.7",      // Parsing de cookies
  "cors": "^2.8.5",               // Cross-Origin Resource Sharing
  "dotenv": "^16.5.0",            // Variáveis de ambiente
  "ejs": "^3.1.9",                // Template engine
  "express": "^4.21.2",           // Framework web
  "express-session": "^1.18.1",  // Gerenciamento de sessões
  "express-validator": "^7.2.1", // Validação de dados
  "helmet": "^8.1.0",             // Middlewares de segurança
  "jsonwebtoken": "^9.0.2",       // JWT tokens
  "mongoose": "^7.5.1"            // ODM MongoDB
}
```

Estrutura de Arquivos

```
backend/
├── config/          # Configurações (DB, JWT, etc.)
├── controllers/     # Lógica de negócio
├── middleware/      # Middlewares customizados
├── models/          # Modelos do banco de dados
├── routes/          # Definição de rotas
├── views/           # Templates EJS
├── public/          # Arquivos estáticos
├── utils/           # Utilitários
├── .env             # Variáveis de ambiente
├── package.json     # Dependências e scripts
└── server.js        # Arquivo principal do servidor
```

Configuração do Ambiente

Variáveis de Ambiente (.env)

```
# Banco de dados MongoDB Atlas
MONGO_URI=mongodb+srv://ygorjs:ygor123@cluster0.lbflaoo.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

# Autenticação JWT
JWT_SECRET=TFThbjhygy454bdd$5656%%$122

# Configurações do servidor
PORT=5000
NODE_ENV=development

# Cookies seguros
COOKIE_SECRET=dfgafJBJB5121%hb*
```

Scripts Disponíveis

```
# Produção
npm start          # Executa com Node.js

# Desenvolvimento
npm run dev        # Executa com Nodemon (hot reload)
```

Configuração de Segurança

- **Helmet.js** - Headers de segurança HTTP
- **CORS** configurado para frontend específico
- **Cookies seguros** com HttpOnly e SameSite
- **Rate limiting** e validação de entrada

- **JWT tokens** com expiração configurável

Estrutura do Projeto

Arquitetura RESTful com Versionamento

A IncubePro implementa uma arquitetura híbrida:

- **API RESTful versionada** (/api/v1/) para integrações
- **Rotas de páginas EJS** (/) para interface web
- **Rotas legacy** para compatibilidade

Padrões de Organização

Controllers

Seguem o padrão de responsabilidade única:

- authController.js - Autenticação (login, register, logout)
- userController.js - Gerenciamento de usuários
- projectController.js - CRUD de projetos
- sessionController.js - Gerenciamento de sessões

Models

Estruturas de dados baseadas em Mongoose schemas:

- User.js - Desenvolvedores, investidores, mentores
- Project.js - Projetos de software
- Session.js - Sessões ativas

Middleware

- auth.js - Autenticação JWT
- requireAuth.js - Proteção de rotas
- validation.js - Validação de dados
- userMiddleware.js - População de dados do usuário

Sistema de Rotas

Rotas de Páginas Web (EJS)

Públicas

GET / # Página inicial

GET /destaques # Cases de sucesso

GET /projetos # Projetos em andamento (pré-login)

GET /login # Formulário de login

GET /register # Formulário de registro

GET /projects/:id # Detalhes de projeto específico

GET /projeto/:id # Alternativa para detalhes do projeto

Protegidas (Requer Autenticação)

GET /profile # Perfil do usuário

GET /meus-projetos # Lista de projetos do usuário

GET /criar-projeto # Formulário de criação de projeto

GET /projects/new # Alternativa para criar projeto

GET /editar-projeto/:id # Edição de projeto (apenas criador)

Funcionalidades Especiais

- **Paginação avançada** nos projetos do usuário (9 por página)
- **Filtros dinâmicos** por status, categoria e busca textual
- **Ordenação personalizada** (título, meta financeira, data)
- **Estatísticas em tempo real** do usuário
- **Validação de permissões** (apenas criador pode editar)

Rotas de Autenticação (Legacy)

POST /api/auth/register # Registro de usuário

POST /api/auth/login # Login de usuário

GET /api/auth/logout # Logout (GET e POST suportados)

Endpoints da API

API v1 - Usuários (/api/v1/users)

Gerais

GET /api/v1/users # Listar usuários (paginado)

POST /api/v1/users # Criar usuário

GET /api/v1/users/:id # Buscar usuário por ID

PUT /api/v1/users/:id # Atualizar usuário (completo)

DELETE /api/v1/users/:id # Deletar usuário

Específicos

GET /api/v1/users/profile # Perfil do usuário logado

PUT /api/v1/users/profile # Atualizar perfil próprio

GET /api/v1/users/:id/stats # Estatísticas do usuário

API v1 - Projetos (/api/v1/projects)

CRUD Completo

GET /api/v1/projects # Listar projetos (paginado, filtros)

POST /api/v1/projects # Criar projeto (apenas developers)

GET /api/v1/projects/:id # Buscar projeto por ID

PUT /api/v1/projects/:id # Atualizar projeto (completo)

PATCH /api/v1/projects/:id # Atualizar projeto (parcial)

DELETE /api/v1/projects/:id # Deletar projeto

Específicos

GET /api/v1/projects/my # Projetos do usuário logado

GET /api/v1/projects/stats # Estatísticas globais de projetos

API v1 - Sessões (/api/v1/sessions)

Gerenciamento de Sessão

POST /api/v1/sessions # Criar sessão (login)

GET /api/v1/sessions/current # Verificar sessão atual

PUT /api/v1/sessions/refresh # Renovar sessão (refresh token)

DELETE /api/v1/sessions # Deletar sessão (logout)

Recursos da API

Paginação Padrão

```
{
  "data": [...],
  "pagination": {
    "current": 1,
    "pages": 5,
    "total": 50,
    "hasNext": true,
    "hasPrev": false
  }
}
```

Filtros Suportados

- **Projetos**: status, categoria, busca textual, ordenação
- **Usuários**: role, status, busca por nome/email
- **Sessões**: histórico de login, dispositivos ativos

Middleware e Segurança

Sistema de Autenticação

JWT (JSON Web Tokens)

- **Tokens seguros** armazenados em cookies HttpOnly
- **Refresh tokens** para renovação automática
- **Expiração configurável** por ambiente
- **Middleware de verificação** em rotas protegidas

Middleware de Autenticação

// Middleware principal de autenticação

```
auth()              # Verifica token JWT válido

// Middlewares específicos
requireAuth()       # Exige usuário logado
requireDeveloper()  # Exige usuário com role 'developer'
populateUser()      # Popula dados do usuário nas views
```

Validação de Dados

Express Validator

- **Validação de entrada** em todos os endpoints
- **Sanitização** de dados do usuário
- **Mensagens de erro** personalizadas
- **Validação de ObjectId** do MongoDB

Middlewares de Validação

```
validateUserRegistration()  # Validação de registro
validateUserLogin()         # Validação de login
validateProjectCreation()   # Validação de criação de projeto
validateProjectUpdate()     # Validação de atualização
validateObjectId()          # Validação de IDs MongoDB
validatePagination()        # Validação de parâmetros de paginação
```

Segurança HTTP

Helmet.js

- **Content Security Policy** (desabilitado para EJS)
- **X-Frame-Options** - Proteção contra clickjacking
- **X-XSS-Protection** - Proteção XSS
- **Strict-Transport-Security** - HTTPS obrigatório em produção

CORS (Cross-Origin Resource Sharing)

```
{
```

```
  origin: process.env.FRONTEND_URL || 'http://localhost:8080',
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With']
}
```

Modelos de Dados

Schema do Usuário (User.js)

Estrutura Principal

```
{
  name: String,            // Nome completo (obrigatório)
  email: String,           // Email único (obrigatório, validado)
  password: String,        // Senha criptografada (min 6 chars)
  role: String,            // 'developer' | 'investor'
  createdAt: Date,         // Data de criação automática
  bio: String,             // Biografia opcional
  skills: [String],        // Array de habilidades
  avatar: String,          // URL do avatar
  projects: [ObjectId]     // Referência aos projetos (ref: Project)
}
```

Validações e Regras

- **Email**: Formato válido, único, convertido para lowercase
- **Senha**: Mínimo 6 caracteres, criptografada com bcrypt (salt 10)
- **Role**: Apenas 'developer' ou 'investor'
- **Skills**: Array de strings para habilidades técnicas
- **Projects**: Relacionamento virtual com projetos criados

Métodos Personalizados

```
// Pré-salvamento: criptografia automática da senha
UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

// Comparação de senhas para login
UserSchema.methods.comparePassword = async function(password) {
  return await bcrypt.compare(password, this.password);
};
```

Schema do Projeto (Project.js)

Estrutura Principal

```
{
  title: String,            // Título do projeto (obrigatório)
  description: String,      // Descrição detalhada (obrigatória)
  status: String,           // Status atual do projeto
  category: String,         // Categoria técnica (obrigatória)
  fundingGoal: Number,      // Meta de financiamento (obrigatória)
  creator: ObjectId,        // Referência ao criador (ref: User)
  rewards: [String],        // Array de recompensas para apoiadores
  currentFunding: Number,   // Valor atual arrecadado (default: 0)
  createdAt: Date           // Data de criação automática
}
```

Enums e Valores Aceitos

Status do Projeto

```
enum: [
```

'Conceito', // Ideia inicial, validação de mercado

'Em andamento', // Desenvolvimento ativo

'Versão beta', // Teste com usuários limitados

'No ar (lançado)' // Produto em produção

]

Categorias Técnicas

enum: [

'Desenvolvimento Web', // Sites, sistemas web

'Aplicativo Mobile', // Apps iOS/Android

'IA/Machine Learning', // Inteligência artificial

'Blockchain', // Tecnologias distribuídas

'IoT', // Internet das Coisas

'Jogos', // Games e entretenimento

'Outros' // Categorias não listadas

]

Validações e Regras

- **Title**: Campo obrigatório com mensagem personalizada
- **Description**: Campo obrigatório para detalhes do projeto
- **FundingGoal**: Valor numérico obrigatório (validação de R$ 100 mínimo no controller)
- **Creator**: Referência obrigatória ao usuário criador
- **CurrentFunding**: Valor padrão 0, atualizado conforme contribuições

---

Controllers - Lógica de Negócio

AuthController (authController.js)

Funcionalidades Principais

- **Registro híbrido** (API JSON + formulário EJS)

- **Login com redirecionamento** inteligente
- **Logout seguro** com limpeza de cookies

Método: register()

// Fluxo de registro

1. Validação de email único

2. Criação do usuário com senha criptografada

3. Resposta diferenciada por tipo de requisição:

   - JSON: Status 201 + mensagem de sucesso

   - EJS: Redirecionamento para /login?registered=true

4. Tratamento de erros com renderização personalizada

Método: login()

// Fluxo de login

1. Validação de credenciais (email + senha)

2. Comparação segura de senha com bcrypt

3. Geração de JWT token (expiração: 2 dias)

4. Resposta diferenciada:

   - JSON: Token + dados do usuário

   - EJS: Cookie HttpOnly + redirecionamento

5. Suporte a parâmetro ?redirect para pós-login

Configuração de Cookies

```
{
  httpOnly: true,           // Previne acesso via JavaScript
  maxAge: 2 * 24 * 60 * 60 * 1000,  // 2 dias em milissegundos
  secure: NODE_ENV === 'production',  // HTTPS apenas em produção
  sameSite: 'strict'        // Proteção CSRF
}
```

ProjectController (projectController.js)

Funcionalidades Avançadas

- **CRUD completo** com validações rigorosas
- **Paginação inteligente** (12 projetos por página)
- **Filtros dinâmicos** (status, categoria, busca textual)
- **Controle de permissões** (apenas criador/admin edita)
- **Estatísticas agregadas** com MongoDB pipelines

Método: createProject()

// Validações rigorosas

- Título: mínimo 3 caracteres

- Descrição: mínimo 10 caracteres

- Meta financeira: mínimo R$ 100

- Role: apenas 'developer' pode criar

- Recompensas: filtro de valores vazios

// Processamento de dados

- Trimming automático de strings

- Conversão de meta para número

- Population automática do criador

- Timestamps automáticos

// Resposta inteligente

- AJAX: JSON com dados do projeto

- Formulário: Redirecionamento para /meus-projetos?created=true

Método: getAllProjects() - Sistema de Filtros

// Filtros suportados

const filters = {};

if (req.query.status) filters.status = req.query.status;

```
if (req.query.category) filters.category = req.query.category;
if (req.query.search) {
  filters.$or = [
    { title: { $regex: req.query.search, $options: 'i' } },
    { description: { $regex: req.query.search, $options: 'i' } }
  ];
}

// Ordenações disponíveis
switch (req.query.sortBy) {
  case 'title': sortBy = { title: 1 }; break;
  case 'fundingGoal': sortBy = { fundingGoal: -1 }; break;
  case 'currentFunding': sortBy = { currentFunding: -1 }; break;
  case 'oldest': sortBy = { createdAt: 1 }; break;
  default: sortBy = { createdAt: -1 }; // Mais recentes primeiro
}
```

Método: getProjectStats() - Análise de Dados

```
// Estatísticas gerais via MongoDB Aggregation
const stats = await Project.aggregate([
  {
    $group: {
      _id: null,
      total: { $sum: 1 },                    // Total de projetos
      totalFunding: { $sum: '$currentFunding' },  // Soma arrecadada
      totalGoal: { $sum: '$fundingGoal' },        // Soma das metas
      avgFunding: { $avg: '$currentFunding' },    // Média arrecadada
      avgGoal: { $avg: '$fundingGoal' }           // Média das metas
```

```
    }
  }
]);

// Distribuição por status e categoria
const statusStats = await Project.aggregate([
  { $group: { _id: '$status', count: { $sum: 1 } } }
]);
```

UserController (userController.js)

Sistema de Permissões

- **Usuário comum**: Apenas próprio perfil
- **Admin**: Acesso total a todos os usuários
- **Developer**: Estatísticas de projetos incluídas

Método: getAllUsers() - Listagem Avançada

```
// Paginação padrão: 10 usuários por página
// Filtros: role, busca por nome/email
// População: projetos relacionados (título, status, categoria)
// Exclusão: campos de senha sempre removidos
// Ordenação: usuários mais recentes primeiro
```

Método: getUserStats() - Dashboard Personalizado

```
// Para developers: estatísticas completas de projetos
stats.developer = {
  totalProjects: projects.length,
  projectsByStatus: {
    conceito: projects.filter(p => p.status === 'Conceito').length,
    emAndamento: projects.filter(p => p.status === 'Em andamento').length,
    beta: projects.filter(p => p.status === 'Versão beta').length,
```

```
    lancado: projects.filter(p => p.status === 'No ar (lançado)').length
  },
  totalFundingGoal: projects.reduce((sum, p) => sum + p.fundingGoal, 0),
  totalCurrentFunding: projects.reduce((sum, p) => sum + p.currentFunding, 0)
};
```

SessionController (sessionController.js)

Gerenciamento Avançado de Sessões

- **Login híbrido** (API + formulário)
- **Verificação de sessão** ativa
- **Refresh token** para renovação
- **Logout seguro** com limpeza completa

Método: createSession() - Login Unificado

```
// Detecção automática do tipo de requisição
const isApiRequest = req.headers['content-type'] === 'application/json' ||
                     req.headers.accept?.includes('application/json');

// Resposta para API
if (isApiRequest) {
  return sendSuccess(res, { token, user: userResponse }, 'Login realizado');
} else {
  // Resposta para formulário EJS
  res.cookie('token', token, cookieOptions);
  return res.redirect(redirect);
}
```

Método: refreshSession() - Renovação Inteligente

```
// Busca usuário atual via middleware auth
// Gera novo token com mesma expiração (2 dias)
```

// Retorna token renovado + dados atualizados do usuário

// Útil para SPAs que precisam manter sessão ativa

Helper de Resposta Padronizada

// Utiliza utils/responseHelper.js para respostas consistentes

sendSuccess(res, data, message, statusCode, meta);

sendError(res, message, errors, statusCode);

sendUnauthorized(res, message);

sendServerError(res, message);

Sistema de Autenticação Detalhado

Fluxo de Autenticação JWT

1. Registro de Usuário

POST /api/auth/register

├── Validação de dados de entrada

├── Verificação de email único

├── Criptografia de senha (bcrypt salt 10)

├── Criação do documento no MongoDB

└── Resposta de sucesso (sem token automático)

2. Login e Geração de Token

POST /api/auth/login

├── Validação de credenciais

├── Comparação segura de senha

├── Geração de JWT token

│ ├── Payload: { id, name, role }

│ ├── Secret: process.env.JWT_SECRET

│ └── Expiração: 2 dias

├── Configuração de cookie HttpOnly

└── Redirecionamento ou resposta JSON

3. Validação de Sessão

Middleware auth() em rotas protegidas

├── Extração do token (cookie ou header)

├── Verificação JWT com secret

├── Decodificação do payload

├── População de req.user

└── Continuação da requisição

Segurança Implementada

Cookies Seguros

```
{
  httpOnly: true,        // Não acessível via JavaScript client-side
  secure: isProduction,  // HTTPS obrigatório em produção
  sameSite: 'strict',    // Proteção contra ataques CSRF
  maxAge: 172800000,     // 2 dias em milissegundos
  path: '/'              // Disponível em toda aplicação
}
```

Proteção de Rotas

- **Middleware auth()**: Validação de token em todas as rotas protegidas
- **Middleware requireAuth()**: Redirecionamento para login se não autenticado
- **Verificação de role**: Controle granular de permissões por função
- **Validação de propriedade**: Usuários só editam próprios recursos

---

Validações e Regras de Negócio

Validações de Usuário

Registro

- **Nome**: Campo obrigatório, string não vazia

- **Email**: Formato válido, único no sistema, convertido para lowercase
- **Senha**: Mínimo 6 caracteres, criptografada automaticamente
- **Role**: Apenas 'developer' ou 'investor' aceitos

Atualização de Perfil

- **Nome**: Opcional, mas não pode ser vazio se fornecido
- **Bio**: String opcional para descrição pessoal
- **Skills**: Array de strings para habilidades técnicas
- **Avatar**: URL opcional para foto de perfil

Validações de Projeto

Criação

```
// Validações obrigatórias
if (!title) errors.push({ field: 'title', message: 'Título é obrigatório' });
if (!description) errors.push({ field: 'description', message: 'Descrição é obrigatória' });
if (!category) errors.push({ field: 'category', message: 'Categoria é obrigatória' });
if (!fundingGoal) errors.push({ field: 'fundingGoal', message: 'Meta de financiamento é obrigatória' });

// Validações de tamanho
if (title.length < 3) {
  errors.push({ field: 'title', message: 'Título deve ter pelo menos 3 caracteres' });
}

if (description.length < 10) {
  errors.push({ field: 'description', message: 'Descrição deve ter pelo menos 10 caracteres' });
}
```

```
// Validações de valor
const fundingGoalNumber = parseFloat(fundingGoal);
if (isNaN(fundingGoalNumber) || fundingGoalNumber < 100) {
  errors.push({ field: 'fundingGoal', message: 'Meta deve ser de pelo menos R$ 100,00' });
}
```

Controle de Permissões

```
// Apenas developers podem criar projetos
if (req.user.role !== 'developer') {
  return res.status(403).json({
    success: false,
    message: 'Apenas desenvolvedores podem criar projetos'
  });
}

// Apenas criador ou admin podem editar/deletar
if (project.creator.toString() !== req.user.id && req.user.role !== 'admin') {
  return res.status(403).json({
    success: false,
    message: 'Você não tem permissão para editar este projeto'
  });
}
```

Tratamento de Erros

Erros de Validação (Mongoose)

```
if (error.name === 'ValidationError') {
  const errors = Object.keys(error.errors).map(key => ({
    field: key,
```

```
    message: error.errors[key].message
  }));

  return res.status(400).json({
    success: false,
    message: 'Erro de validação',
    errors
  });
}
```

Erros de Cast (MongoDB ObjectId)

```
if (error.name === 'CastError') {
  return res.status(400).json({
    success: false,
    message: 'ID do projeto inválido'
  });
}
```

Sistema de Middlewares Detalhado

Middleware de Autenticação (auth.js)

Funcionalidades Principais

- **Autenticação JWT dupla** (API e páginas EJS)
- **Verificação de tokens** via header Authorization e cookies
- **Tratamento diferenciado** por tipo de requisição
- **Limpeza automática** de tokens inválidos

Método: auth() - Autenticação API

```
// Fluxo de autenticação para API
1. Extração do token:
   - Header "Authorization: Bearer [token]"
```

- Cookie "token" como fallback

2. Verificação JWT com process.env.JWT_SECRET

3. Busca do usuário no MongoDB (sem senha)

4. População de req.user para controllers

5. Tratamento específico de erros JWT

Método: authPage() - Autenticação EJS

// Fluxo de autenticação para páginas

1. Verificação exclusiva via cookie "token"

2. Redirecionamento automático para /login se não autenticado

3. População simultânea:

- req.user para controllers

- res.locals.user para views EJS

4. Limpeza de cookies inválidos

5. Redirecionamentos com parâmetros de erro

Tratamento de Erros Específicos

// Erros JWT tratados individualmente

JsonWebTokenError: "Token inválido"

TokenExpiredError: "Token expirado"

ValidationError: "Dados inválidos"

UserNotFound: Limpeza automática de cookie + redirecionamento

Middleware de Proteção (requireAuth.js)

Sistema de Proteção Avançado

- **Detecção automática** de tipo de requisição (AJAX vs página)
- **Redirecionamento inteligente** com preservação da URL original
- **Limpeza proativa** de cookies inválidos
- **Controle granular** de roles e permissões

Método: requireAuth() - Proteção Principal

// Fluxo de proteção de rotas

1. Verificação de token em cookies

2. Decodificação e validação JWT

3. Busca e validação do usuário no banco

4. População dupla: req.user + res.locals.user

5. Resposta diferenciada:

   - AJAX: JSON com erro 401/404

   - Página: Redirecionamento para /login?redirect=url_original

Sistema de Roles Hierárquico

// Middleware requireRole(roles)

- Suporte a role única ou array de roles

- Verificação de req.user.role

- Mensagens de erro personalizadas por contexto

- Renderização de página de erro 403 para EJS

// Middlewares específicos

requireDeveloper() # Apenas role 'developer'

requireInvestor()   # Apenas role 'investor'

requireAnyRole()   # Qualquer usuário logado

Configuração de Cookies Seguros

```
{
  httpOnly: true,                    // Não acessível via JavaScript
  secure: process.env.NODE_ENV === 'production',  // HTTPS em produção
  sameSite: 'strict',
```

Configuração de Cookies Seguros

{

```
  httpOnly: true,                    // Não acessível via JavaScript
  secure: process.env.NODE_ENV === 'production',  // HTTPS em produção
  sameSite: 'strict',                // Proteção CSRF
  path: '/'                          // Disponível em toda aplicação
}
```

Middleware de População (userMiddleware.js)

População Não-Obstrutiva de Usuário

- **Verificação opcional** sem bloquear acesso
- **População automática** para views EJS
- **Limpeza silenciosa** de tokens inválidos
- **Fallback gracioso** para usuários não logados

Método: populateUser() - População Inteligente

// Fluxo de população opcional

1. Verificação não-obstrutiva de token cookie

2. Validação JWT sem gerar erro

3. Busca de usuário atualizado no banco

4. População condicional:

   - Token válido: req.user + res.locals.user = userData

   - Token inválido: req.user + res.locals.user = null

5. Limpeza silenciosa de cookies inválidos

6. Continuação da requisição sempre (next())

Casos de Uso Principais

// Páginas públicas com contexto de usuário

- Página inicial com dados do usuário se logado

- Navegação com menu personalizado

- Funcionalidades condicionais baseadas em login

- Analytics de usuário para páginas públicas

Sistema de Validação (validation.js)

Validação Robusta com Express-Validator

- **Validação em camadas** (sanitização + validação + formatação)
- **Mensagens personalizadas** em português
- **Tratamento unificado** de erros de validação
- **Suporte completo** a arrays e objetos aninhados

Estrutura de Validação

// Padrão de validação em 3 etapas

1. Definição de regras de validação

2. Processamento via express-validator

3. Tratamento de erros via handleValidationErrors()

Validações de Usuário

validateUserRegistration

// Validação completa de registro

- name: 2-50 chars, trim automático

- email: formato válido, normalização lowercase

- password: min 6 chars + complexidade (maiúscula, minúscula, número)

- role: enum ['developer', 'investor']

validateUserUpdate

// Validação de atualização (campos opcionais)

- name: 2-50 chars se fornecido

- bio: máximo 500 chars

- skills: array de strings (1-30 chars cada)

- avatar: URL válida (se implementado)

Validações de Projeto

validateProjectCreation

// Validação rigorosa de criação

- title: 3-100 chars obrigatório

- description: 10-2000 chars obrigatório

- category: enum de categorias técnicas

- status: enum de status do projeto (opcional)

- fundingGoal: número > 0 obrigatório

- rewards: array opcional de strings (1-200 chars)

validateProjectUpdate

// Validação de atualização (tudo opcional)

- Mesmas regras da criação

- Todos os campos opcionais

- Mantém validação de tipos e ranges

- Permite atualizações parciais

Validações de Parâmetros

validateObjectId(paramName)

// Validação de IDs MongoDB

- Verifica formato ObjectId válido

- Parâmetro customizável (id, userId, projectId)

- Erro específico para IDs inválidos

validatePagination

// Query parameters para listagens

- page: inteiro >= 1

- limit: inteiro entre 1-100

- category: enum de categorias válidas

- status: enum de status válidos

- search: string livre para busca textual

Sistema de Tratamento de Erros

// handleValidationErrors() - Middleware unificado

```
const handleValidationErrors = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    const formattedErrors = errors.array().map(error => ({
      field: error.path,        // Campo com erro
      message: error.msg,       // Mensagem em português
      value: error.value        // Valor que causou erro
    }));
    return sendValidationError(res, formattedErrors);
  }
  next();
};
```

Validações Customizadas

```
// Exemplo: validação de complexidade de senha
body('password')
  .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/)
  .withMessage('Senha deve conter pelo menos uma letra minúscula, uma maiúscula e um número')

// Exemplo: validação de valor monetário
body('fundingGoal')
  .custom((value) => {
    if (value <= 0) {
      throw new Error('Meta de financiamento deve ser maior que zero');
    }
    return true;
  })
```

Integração entre Middlewares

Cadeia de Middleware Típica

// Para rotas de API protegidas

[validateObjectId('id'), auth, validateProjectUpdate, projectController.updateProject]

// Para páginas EJS protegidas

[requireAuth, populateUser, pageController.renderUserProfile]

// Para páginas públicas com contexto

[populateUser, pageController.renderHomePage]

// Para rotas com controle de role

[requireAuth, requireDeveloper, validateProjectCreation, projectController.createProject]

Fluxo de Dados entre Middlewares

// Sequência de enriquecimento da requisição

1. validation.js -> Sanitiza e valida dados de entrada

2. auth.js/requireAuth.js -> Autentica e popula req.user

3. userMiddleware.js -> Enriquece res.locals.user para views

4. Controller -> Processa lógica de negócio com dados validados

---

Configurações e Infraestrutura

Configuração do Banco de Dados (config/db.js)

Conexão Robusta com MongoDB Atlas

A IncubePro utiliza uma configuração avançada de conexão com MongoDB Atlas, implementando práticas de resiliência e monitoramento.

Funcionalidades Principais

- **Conexão resiliente** com retry automático
- **Event listeners** para monitoramento de estado
- **Encerramento limpo** com tratamento de sinais
- **Logs detalhados** para debugging e produção

Estrutura da Configuração

```
// Opções de conexão optimizadas
const connectOptions = {
  retryWrites: true,        // Retry automático em operações de escrita
  w: 'majority'             // Write concern para consistência
};

// Configuração de conexão principal
const conn = await mongoose.connect(process.env.MONGO_URI, connectOptions);
```

Sistema de Monitoramento Avançado

Event Listeners de Conexão

```
// Monitoramento de erros de conexão
mongoose.connection.on('error', (err) => {
  console.error('Erro na conexão MongoDB:', err);
  // TODO: Implementar sistema de notificação/alertas
});

// Monitoramento de desconexões
mongoose.connection.on('disconnected', () => {
  console.warn('Desconectado do MongoDB');
  // TODO: Implementar tentativas de reconexão
});
```

```
// Sucesso na conexão
console.log('Conectado ao MongoDB com sucesso!!!');
```

Encerramento Gracioso

```
// Tratamento de SIGINT (Ctrl+C)
process.on('SIGINT', async () => {
  await mongoose.connection.close();
  console.log('Conexão com MongoDB encerrada devido ao encerramento da aplicação');
  process.exit(0);
});
```

Tratamento Robusto de Erros

- **Erro de conexão**: Log detalhado + exit(1) para falha crítica
- **Timeout de conexão**: Configuração implícita via MongoDB driver
- **Perda de conexão**: Event listener para logging (reconexão automática via Mongoose)
- **Encerramento da aplicação**: Fechamento limpo da conexão

Configurações de Produção Recomendadas

```
// Variáveis de ambiente para produção
MONGO_URI=mongodb+srv://user:pass@cluster.mongodb.net/dbname?retryWrites=true&w=majority&serverSelectionTimeoutMS=5000&connectTimeoutMS=10000&authSource=admin&replicaSet=atlas-replica-set

// Opções adicionais para produção (futuras implementações)
const productionOptions = {
  maxPoolSize: 10,          // Limite de conexões simultâneas
  serverSelectionTimeoutMS: 5000,  // Timeout para seleção de servidor
  socketTimeoutMS: 45000,   // Timeout para operações socket
  family: 4                 // IPv4 apenas
```

};

Sistema de Respostas Padronizadas (utils/responseHelper.js)

Padronização Completa de Respostas API

O responseHelper.js garante consistência total nas respostas da API, implementando padrões REST e facilitando o desenvolvimento frontend.

Estrutura de Resposta Padrão

```
// Resposta de sucesso
{
  "success": true,
  "message": "Descrição da operação",
  "data": { /* dados retornados */ },
  "meta": { /* metadados opcionais */ }
}

// Resposta de erro
{
  "success": false,
  "message": "Descrição do erro",
  "errors": [ /* array de erros detalhados */ ]
}
```

Métodos de Sucesso

sendSuccess(res, data, message, statusCode, meta)

```
// Método mais flexível para respostas de sucesso
// Usado em: listagens, atualizações, consultas

// Exemplo de uso
```

```
sendSuccess(res, users, 'Usuários listados com sucesso', 200, {
  total: 50,
  page: 1,
  limit: 10
});

// Resposta gerada
{
  "success": true,
  "message": "Usuários listados com sucesso",
  "data": [...users],
  "meta": {
    "total": 50,
    "page": 1,
    "limit": 10
  }
}
```

sendCreated(res, data, message)

```
// Especializado para recursos criados (HTTP 201)
// Usado em: registro de usuários, criação de projetos

sendCreated(res, newProject, 'Projeto criado com sucesso');
// Status: 201 Created
```

Métodos de Erro

sendError(res, message, errors, statusCode)

```
// Método genérico para erros
// Usado em: validações, erros de negócio, conflitos
```

```
sendError(res, 'Dados inválidos', [
  { field: 'email', message: 'Email já existe' }
], 409);
```

sendValidationError(res, errors, message)

```
// Especializado para erros de validação (HTTP 422)
// Usado com: express-validator, validações customizadas

sendValidationError(res, [
  { field: 'name', message: 'Nome é obrigatório' },
  { field: 'email', message: 'Email deve ter formato válido' }
]);
```

Métodos de Status Específicos

```
sendNotFound(res, 'Usuário não encontrado');        // 404
sendUnauthorized(res, 'Token inválido');            // 401
sendServerError(res, 'Erro interno do servidor');   // 500
```

Integração com Controllers

```
// Exemplo: UserController usando responseHelper
const { sendSuccess, sendError, sendCreated } = require('../utils/responseHelper');

// Criação de usuário
try {
  const user = await User.create(userData);
  sendCreated(res, user, 'Usuário criado com sucesso');
} catch (error) {
  if (error.code === 11000) {
    sendError(res, 'Email já existe', null, 409);
```

```
  } else {
    sendServerError(res, 'Erro ao criar usuário');
  }
}
```

Benefícios da Padronização

- **Frontend consistente**: Tratamento uniforme de respostas
- **Debugging facilitado**: Estrutura previsível para logs
- **Documentação automática**: Padrão claro para API docs
- **Manutenibilidade**: Mudanças centralizadas em um arquivo
- **Testes unitários**: Assertions padronizadas
-

Configuração Principal do Servidor (server.js)

Arquitetura Híbrida Avançada

O server.js implementa uma arquitetura híbrida que suporta simultaneamente:

- **API RESTful versionada** para integrações externas
- **Interface web com EJS** para usuários finais
- **Compatibilidade legacy** para transições suaves

Stack de Middlewares Avançados

Segurança com Helmet.js

```
// Configuração de segurança HTTP
app.use(helmet({
  contentSecurityPolicy: false  // Desabilitado para compatibilidade com EJS
}));

// Headers de segurança aplicados automaticamente:
// - X-DNS-Prefetch-Control
// - X-Frame-Options: DENY
```

```
// - X-Download-Options: noopen
// - X-Content-Type-Options: nosniff
// - X-XSS-Protection: 1; mode=block
```

CORS Configurado para Produção

```
const corsOptions = {
  origin: process.env.FRONTEND_URL || 'http://localhost:8080',
  credentials: true,                    // Suporte a cookies de autenticação
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With']
};
```

Parsing Robusto de Dados

```
// Limites de payload para segurança
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true, limit: '10mb' }));

// Cookie parser com secret para assinatura
app.use(cookieParser(process.env.COOKIE_SECRET));
```

Sistema de Debug Avançado

Debug de Arquivos Estáticos

```
// Verificação de existência da pasta public
const publicPath = path.join(__dirname, 'public');
console.log('🔍 Caminho da pasta public:', publicPath);
console.log('📁 Pasta public existe?', fs.existsSync(publicPath));

// Listagem automática de arquivos
if (fs.existsSync(publicPath)) {
```

```
  const files = fs.readdirSync(publicPath);
  console.log('📄 Arquivos na pasta public:', files);
}
```

Endpoints de Debug

```
// GET /debug/public - Lista arquivos estáticos
app.get('/debug/public', (req, res) => {
  const files = fs.readdirSync(publicPath);
  const fileDetails = files.map(file => ({
    name: file,
    size: fs.statSync(path.join(publicPath, file)).size,
    isFile: fs.statSync(path.join(publicPath, file)).isFile(),
    path: `/public/${file}`,
    directPath: `/${file}`
  }));

  res.json({ publicPath, files: fileDetails });
});

// GET /debug/user - Verifica usuário logado
app.get('/debug/user', (req, res) => {
  res.json({
    user: req.user || null,
    hasToken: !!req.cookies.token,
    token: req.cookies.token ? 'presente' : 'ausente'
  });
});
```

Arquitetura de Rotas Hierárquica

Rotas de Páginas (EJS) - Sem Versionamento

```
// Páginas para usuários finais
app.use('/', pageRoutes);  // /, /login, /profile, /projetos, etc.
```

API v1 - Com Versionamento

```
// API principal versionada
app.use('/api/v1/users', userRoutes);
app.use('/api/v1/projects', projectRoutes);
app.use('/api/v1/sessions', sessionRoutes);

// Health check para monitoramento
app.get('/api/health', (req, res) => {
  res.json({
    status: 'OK',
    message: 'IncubePro API está funcionando!',
    version: '1.0.0',
    timestamp: new Date().toISOString()
  });
});
```

Rotas Legacy e Aliases

```
// Compatibilidade com versões anteriores
app.use('/api/auth', authRoutes);  // Rotas antigas de autenticação

// Alias da API v1 como padrão (para facilitar desenvolvimento)
app.use('/api/users', userRoutes);
app.use('/api/projects', projectRoutes);
app.use('/api/sessions', sessionRoutes);
```

Sistema de Tratamento de Erros Avançado

Error Handling Diferenciado

```
// Middleware para rotas de API não encontradas
app.use('/api/*', (req, res) => {
  res.status(404).json({
    success: false,
    message: 'Endpoint da API não encontrado',
    availableVersions: ['v1'],
    availableEndpoints: {
      v1: [
        'GET /api/v1/users - Listar usuários',
        'POST /api/v1/users - Criar usuário',
        'GET /api/v1/projects - Listar projetos',
        'POST /api/v1/projects - Criar projeto',
        // ... lista completa de endpoints
      ]
    }
  });
});

// 404 para páginas EJS
app.use((req, res) => {
  res.status(404).render('404', { title: 'Página não encontrada' });
});
```

Error Handler Global

```
app.use((err, req, res, next) => {
  console.error('Erro não tratado:', err);
```

```
  // Diferenciação de resposta por tipo de requisição
  if (req.originalUrl.startsWith('/api/')) {
    // Resposta JSON para API
    return res.status(500).json({
      success: false,
      message: 'Erro interno do servidor',
      ...(process.env.NODE_ENV === 'development' && { error: err.message })
    });
  }

  // Página de erro para interface web
  res.status(500).render('error', {
    title: 'Erro interno',
    error: process.env.NODE_ENV === 'development' ? err : null
  });
});
```

Inicialização Robusta do Servidor

Startup Sequence

```
const startServer = async () => {
  try {
    // 1. Conectar ao banco de dados
    await connectDB();

    // 2. Iniciar servidor HTTP
    app.listen(PORT, () => {
      console.log('🚀 IncubePro API iniciada com sucesso!');
```

```
      console.log(` 📡 Servidor rodando em: http://localhost:${PORT}`);

      // Logs detalhados em desenvolvimento
      if (process.env.NODE_ENV === 'development') {
        logAvailableEndpoints();
      }
    });
  } catch (err) {
    console.error('❌ Erro ao iniciar o servidor:', err.message);
    process.exit(1);  // Falha crítica
  }
};
```

Graceful Shutdown

```
// Tratamento de sinais para encerramento limpo
process.on('SIGTERM', () => {
  console.log('🛑 SIGTERM recebido, encerrando servidor...');
  // TODO: Implementar encerramento limpo de conexões ativas
  process.exit(0);
});

process.on('SIGINT', () => {
  console.log('🛑 SIGINT recebido, encerrando servidor...');
  process.exit(0);
});
```

Logging e Monitoramento

Middleware de Logging em Desenvolvimento

```
if (process.env.NODE_ENV === 'development') {
  app.use((req, res, next) => {
    const timestamp = new Date().toISOString();
    console.log(`${req.method} ${req.originalUrl} - ${timestamp}`);
    next();
  });
}
```

Listagem Automática de Endpoints

```
const logAvailableEndpoints = () => {
  console.log('   Endpoints principais:');
  console.log('   GET  /api/v1/users          - Listar usuários');
  console.log('   POST /api/v1/users          - Criar usuário');
  console.log('   GET  /api/v1/projects       - Listar projetos');
  console.log('   POST /api/v1/projects       - Criar projeto');
  console.log('   POST /api/v1/sessions       - Login');
  console.log('   GET  /api/v1/sessions/current - Verificar sessão');
  console.log('   GET  /api/health            - Health check');
  console.log('   GET  /debug/public          - Debug arquivos públicos');
  console.log('   GET  /debug/user            - Debug usuário logado');
};
```